

15.2

styled-components

styled-components는 CSS 문법을 그대로 사용하면서 결과물을 스타일링된 **styled** 컴포넌트 component 형태로 만들어주는 오픈소스 라이브러리입니다. 컴포넌트 개념을 사용하기 때문에 리액트와 굉장히 궁합이 잘 맞으며 전 세계적으로 리액트 개발에 많이 사용되고 있습니다. 마지막 장에서 미니 프로젝트를 진행할 때 **styled-components**를 사용할 예정이기 때문에 간단하게 배워봅시다.

1 styled-components 설치하기

styled-components를 사용하기 위해서 프로젝트에 설치를 해줘야 합니다. 아래 명령어를 통해서 최신 버전의 **styled-components**를 설치할 수 있습니다.

npm을 사용하는 경우

```
npm install --save styled-components
```

yarn을 사용하는 경우

```
yarn add styled-components
```

설치가 끝났다면 리액트 프로젝트에서 아래처럼 **MainPage**라는 이름의 간단한 컴포넌트를 하나 만들어 실제로 **styled-components**가 잘 돌아가는지 확인할 수 있습니다.

```
01 import React from "react";
02 import styled from 'styled-components';
03
04 const Wrapper = styled.div`
```

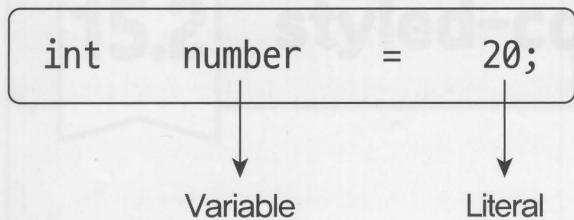
```
05     padding: 1em;
06     background: grey;
07   ;
08
09 const Title = styled.h1`  
10   font-size: 1.5em;  
11   color: white;  
12   text-align: center;  
13 `;
14
15 function MainPage(props) {  
16   return (  
17     <Wrapper>  
18       <Title>  
19         안녕, 리액트!  
20       </Title>  
21     </Wrapper>  
22   )  
23 }  
24
25 export default MainPage;
```

스타일이 잘 적용되었다면 회색 배경에 흰색 글씨로 ‘안녕, 리액트!’라는 문자열이 출력되는 것을 볼 수 있습니다.

2 styled-components 기본 사용법

styled-components는 태그드 템플릿 리터럴tagged template literal을 사용하여 구성 요소의 스타일을 지정합니다. 여기에서 템플릿 리터럴template literal이라는 것이 등장하는데 이것은 자바스크립트에서 제공하는 문법 중 하나입니다. 템플릿 리터럴에 대해 설명하기 전에 먼저 리터럴literal에 대해 알아야 합니다. 프로그래밍에서 리터럴은 소스코드의 고정된 값을 의미합니다. 흔히 상수constant와 혼갈려 하는 경우가 있는데 둘은 다른 개념입니다. 다음 그림을 봅시다.

Literals in Java



▶ 자바의 리터럴

위의 그림을 보면 대입 연산자(=)의 왼쪽에는 `number`라는 이름의 변수^{variable}가 등장하고 오른쪽에는 정수 20이 등장합니다. 여기에서 오른쪽에 있는 정수 20이 바로 리터럴입니다. 소스코드 상에 있는 고정된 값을 의미하는 것이죠. 다른 예제 코드를 살펴봅시다.

```
01 // 정수 리터럴 (Integer literal)
02 const myNumber = 10;
03
04 // 문자열 리터럴 (String literal)
05 const myStr = 'Hello';
06
07 // 배열 리터럴 (Array literal)
08 const myArray = [];
09
10 // 객체 리터럴 (Object literal)
11 const myObject = {};
```

위 코드에는 여러 가지 종류의 리터럴이 나와 있습니다. 또한 변수를 선언할 때 `var`나 `let`을 사용하지 않고 상수를 의미하는 `const`를 사용했는데, 이렇게 선언하게 되면 해당 변수들이 모두 상수가 됩니다. 상수는 변하지 않는 수를 의미하는데 한 번 선언된 이후에는 값을 바꿀 수 없는 것이죠. 그리고 대입 연산자(=)의 오른쪽에 있는 값이 모두 리터럴이 됩니다.

그렇다면 템플릿 리터럴은 무엇일까요? 템플릿 리터럴은 말 그대로 리터럴을 템플릿 형태로 사용하는 자바스크립트의 문법인데, backticks(`)를 사용하여 문자열을 작성하고 그 안에 대체 가능한 `expression`을 넣는 방법입니다. 여기에서 이 `expression`을 대체라는 뜻을 가진

substitution이라고 부릅니다. 템플릿 리터럴은 또 크게 언태그드 템플릿 리터럴 Untagged template literal과 태그드 템플릿 리터럴로 나뉩니다. 아래 예제 코드를 보도록 합시다.

```
01 // Untagged template literal
02 // 단순한 문자열
03 `string text`
04
05 // 여러 줄(Multi-line)에 걸친 문자열
06 `string text line 1
07 string text line 2`
08
09 // 대체 가능한 expression이 들어있는 문자열
10 `string text ${expression} string text`
11
12
13 // Tagged template literal
14 // myFunction의 파라미터로 expression으로 구분된 문자열 배열과 expression이
    순서대로 들어간 형태로 호출됨
15 myFunction`string text ${expression} string text`;
```

위 코드에 나오는 것처럼 언태그드 템플릿 리터럴은 보통 문자열을 여러 줄에 걸쳐서 작성하거나 포매팅 formatting을 하기 위해서 사용합니다. 태그드 템플릿 리터럴은 앞에 나와있는 태그 함수 tag function를 호출하여 결과를 리턴합니다. 여기에서 태그 함수의 파라미터는 expression으로 구분된 문자열 배열과 expression이 순서대로 들어가게 됩니다. 좀 더 쉬운 이해를 위해 아래 예제 코드를 봅시다.

```
01 const name = '인제';
02 const region = '서울';
03
04 function myTagFunction(strings, nameExp, regionExp) {
05     let str0 = strings[0]; // "제 이름은 "
06     let str1 = strings[1]; // "이고, 사는 곳은 "
07     let str2 = strings[2]; // "입니다."
```

```
08
09      // 여기에서도 template literal을 사용하여 리턴할 수 있음
10      return `${str0}${nameExp}${str1}${regionExp}${str2}`;
11  }
12
13 const output = myTagFunction`제 이름은 ${name}이고, 사는 곳은 ${region}입니다.`;
14
15 // 출력 결과
16 // 제 이름은 인제이고, 사는 곳은 서울입니다.
17 console.log(output);
```

위 코드는 태그드 템플릿 리터럴을 사용한 예제입니다. 태그 함수에 파라미터가 어떻게 들어가는지 쉽게 파악되죠? 이처럼 태그드 템플릿 리터럴을 사용하면 문자열과 expression을 태그 함수의 파라미터로 넣어 호출한 결과를 받게 됩니다. 더 자세한 내용이 궁금하면 아래 링크를 참고하세요.

☞ https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals

styled-components는 태그드 템플릿 리터럴을 사용하여 CSS 속성이 적용된 리액트 컴포넌트를 만들어 줍니다. **styled-components**를 사용하는 기본적인 방법은 아래와 같이 backticks (`)로 둘러싸인 문자열 부분에 CSS 속성을 넣고 태그 함수 위치에는 **styled**.<HTML 태그> 형태로 사용합니다. 이렇게 하면 해당 HTML 태그에 CSS 속성들이 적용된 형태의 리액트 컴포넌트가 만들어집니다.

```
01 import React from "react";
02 import styled from "styled-components";
03
04 const Wrapper = styled.div`
05   padding: 1em;
06   background: grey;
07 `;
```

3 styled-components의 props 사용하기

styled-components에서는 조건이나 동적으로 변하는 값을 사용해서 스타일링할 수는 없을까요? 물론 가능합니다. 이것을 위해 제공하는 기능이 바로 `props`입니다. 리액트 컴포넌트의 `props`와 같은 개념으로 이해하면 되는데 예제 코드를 한번 볼까요?

```
01 import React from "react";
02 import styled from "styled-components";
03
04 const Button = styled.button`  

05   color: ${props => props.dark ? "white" : "dark"};  

06   background: ${props => props.dark ? "black" : "white"};  

07   border: 1px solid black;
08 `;
09
10 function Sample(props) {
11   return (
12     <div>
13       <Button>Normal</Button>
14       <Button dark>Dark</Button>
15     </div>
16   )
17 }
18
19 export default Sample;
```

위의 코드에는 `Button`이라는 컴포넌트가 등장합니다. 이 컴포넌트는 `styled-components`를 사용해서 만들어진 것입니다. 그리고 `styled-components`를 사용하는 부분의 CSS 속성을 보면 내부에 `props`가 사용된 것을 볼 수 있습니다. 여기에서의 `props`는 해당 컴포넌트에 사용된 `props`를 의미합니다. 따라서 실제 `Button` 컴포넌트를 사용하는 부분의 코드를 보면 `<Button dark>Dark</Button>`처럼 `props`로 `dark`를 넣어 주는 것을 볼 수 있습니다. 그리고 이렇게 들어간 `props`는 그대로 `styled-components`로 전달됩니다. 이 기능을 사용하면 `styled-components`를 사용하여 다양한 스타일을 자유자재로 구현할 수 있습니다.

4 styled-components의 스타일 확장하기

앞에서 styled-components를 사용하면 리액트 컴포넌트가 생성된다고 설명했습니다. 그렇다면 이렇게 생성된 컴포넌트를 기반으로 추가적인 스타일을 적용하고 싶을 경우에는 어떻게 해야 할까요? styled-components에서는 이를 위한 스타일 확장 기능을 제공합니다. 아래 예제 코드를 봅시다.

```
01 import React from "react";
02 import styled from 'styled-components';
03
04 // Button 컴포넌트
05 const Button = styled.button`color: grey;
06     border: 2px solid palevioletred;
07 `;
08
09
10 // Button에 style이 추가된 RoundedButton 컴포넌트
11 const RoundedButton = styled(Button)`border-radius: 16px;
12 `;
13
14
15 function Sample(props) {
16     return (
17         <div>
18             <Button>Normal</Button>
19             <RoundedButton>Rounded</RoundedButton>
20         </div>
21     )
22 }
23
24 export default Sample;
```

위의 코드를 보면 Button 컴포넌트와 RoundedButton 컴포넌트가 나옵니다. 먼저 Button 컴포넌트는 HTML의 button 태그를 기반으로 만들어진 단순한 버튼입니다. 그리고 Rounded

`Button` 컴포넌트를 만드는 부분을 자세히 보면 HTML 태그가 빠져있고 `Button` 컴포넌트가 팔호로 둘러싸인 채로 들어가 있는 것을 볼 수 있습니다. 이것이 바로 다른 컴포넌트의 스타일을 확장해서 사용하는 부분입니다. `RoundedButton` 컴포넌트는 `Button` 컴포넌트에서 모서리를 둥글게 만든 컴포넌트입니다.

더 다양한 형태로 스타일을 확장할 수 있지만 이 책에서는 기본적인 내용만을 다뤘습니다. 또한 이 책에서 다루지 않은 `styled-components`의 기능이 굉장히 많기 때문에 관심이 있는 독자는 아래 공식 문서를 통해 추가로 학습하기 바랍니다.

☞ <https://styled-components.com/docs>

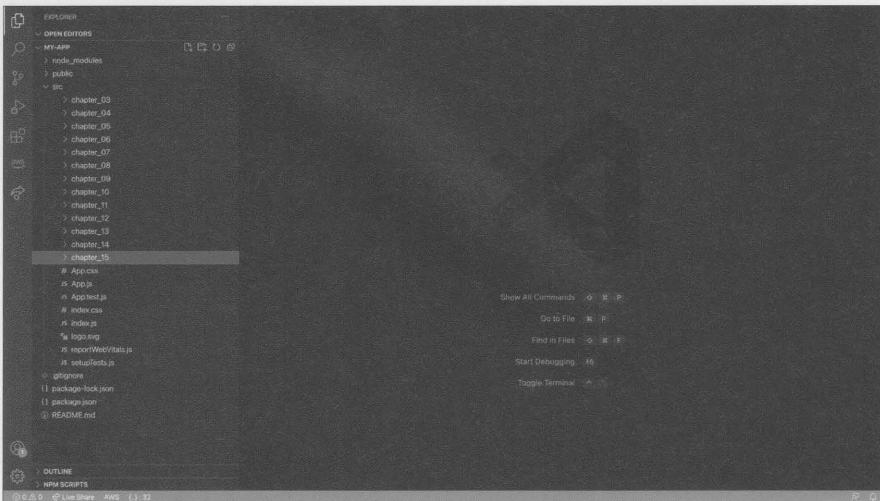
PRACTICE

15.3

실습

styled-components를 사용하여 스타일링해 보기

이번 실습에서는 CSS와 styled-components를 사용하여 직접 컴포넌트를 스타일링해 보도록 하겠습니다. 먼저 VS Code로 앞에서 create-react-app을 이용해 만든 프로젝트를 엽니다. 그리고 아래와 같이 chapter_15라는 이름으로 폴더를 하나 생성합니다.



▶ 실습 화면 01

그다음 만든 폴더에 `Blocks.jsx`라는 이름의 파일을 새로 만듭니다. 코드를 작성하기 전에 먼저 styled-components 패키지를 설치하겠습니다. VS Code의 상단 메뉴에서 Terminal > New Terminal을 눌러 새로운 터미널을 하나 실행시킨 뒤 아래 명령어를 실행합니다.

```
$ npm install --save styled-components
```

`npm install` 명령어는 npm이라는 패키지 관리 서비스에 등록되어 있는 패키지를 설치할 수 있게 해주는 명령어입니다. 이 책의 0장인 ‘준비하기’에서 Node.js를 설치할 때 함께 설치된 것입니다. 위 명령어를 실행하면 `styled-components` 패키지가 아래 그림과 같이 설치됩니다.

The screenshot shows the VS Code interface. In the Explorer sidebar, there is a tree view of a project structure. At the top level, there is a folder named 'Blocks' which contains files like 'App.css', 'App.js', 'App.test.js', 'index.css', 'index.html', 'log.svg', 'reporterVitals.js', and 'secrets.js'. Below 'Blocks' are subfolders for chapters 03 through 15, each containing its own files. The 'node_modules' folder is also visible. In the center, the code editor has a file named 'Blocks.jsx' open. In the bottom right corner, the Terminal tab is active, displaying the output of the npm install command. The output shows that 13 packages were added and 1407 packages were audited. It also mentions 169 packages looking for funding and 9 moderate severity vulnerabilities. The terminal prompt ends with '(base) soulgogolp-MacBook-Pro:my-apps'.

▶ 실습 화면 02

`styled-components` 패키지 설치가 끝났으면 아래 코드처럼 `Blocks`라는 이름의 함수 컴포넌트를 만듭니다. `Blocks` 컴포넌트는 실제로 `styled-components`를 사용하여 스타일링하는 컴포넌트입니다.

```
01 import styled from "styled-components";
02
03 const Wrapper = styled.div`  
04   padding: 1rem;  
05   display: flex;  
06   flex-direction: row;  
07   align-items: flex-start;
```

```
08
09     justify-content: flex-start;
10     background-color: lightgrey;
11   ;
12
13 const Block = styled.div`  

14   padding: ${props => props.padding};  

15   border: 1px solid black;  

16   border-radius: 1rem;  

17   background-color: ${props => props.backgroundColor};  

18   color: white;  

19   font-size: 2rem;  

20   font-weight: bold;  

21   text-align: center;  

22 `;  

23
24 const blockItems = [  

25   {
26     label: "1",
27     padding: "1rem",
28     backgroundColor: "red",
29   },
30   {
31     label: "2",
32     padding: "3rem",
33     backgroundColor: "green",
34   },
35   {
36     label: "3",
37     padding: "2rem",
38     backgroundColor: "blue",
39   },
40 ];
```

```

41  function Blocks(props) {
42      return (
43          <Wrapper>
44              {blockItems.map((blockItem) => {
45                  return (
46                      <Block
47                          padding={blockItem.padding}
48                          backgroundColor={blockItem.backgroundColor}
49                      >
50                          {blockItem.label}
51                      </Block>
52                  );
53              })}
54          </Wrapper>
55      );
56  }
57
58  export default Blocks;

```

The screenshot shows a code editor with the following details:

- File Structure:**
 - OPEN EDITORS: Shows 'Blocks.jsx' and 'index.html'.
 - MY APP:
 - node_modules
 - public
 - chapter_08
 - chapter_09
 - chapter_10
 - chapter_11
 - chapter_12
 - chapter_13
 - chapter_14
 - chapter_15
 - Blocks.jsx
 - # App.css
 - # App.js
 - # App.test.js
 - # index.css
 - # logo.svg
 - # package.json
 - # package-lock.json
 - # README.md
 - gitignore
 - NPM Scripts
- Code Content:** The 'Blocks.jsx' file contains the code from the previous snippet.
- Bottom Status Bar:** Shows line 16, column 1, spaces: 4, UTF-8, JS, JavaScript React, and Preter.

▶ 실습 화면 03

```
blocks.js
  1  import React from 'react';
  2  import { Block } from './Block';
  3
  4  const blockItems = [
  5    {
  6      label: "1",
  7      padding: "1rem",
  8      backgroundColor: "red",
  9    },
 10    {
 11      label: "2",
 12      padding: "3rem",
 13      backgroundColor: "green",
 14    },
 15    {
 16      label: "3",
 17      padding: "2rem",
 18      backgroundColor: "blue",
 19    },
 20  ];
 21
 22  function Blocks(props) {
 23    return (
 24      <Wrapper>
 25        <blockItems.map(blockItem) => {
 26          return (
 27            <Block
 28              padding={blockItem.padding}
 29              backgroundColor={blockItem.backgroundColor}
 30            >
 31              {blockItem.label}
 32            </Block>
 33          );
 34        }
 35      </Wrapper>
 36    );
 37  }
 38
 39  export default Blocks;
```

▶ 실습 화면 04

앞의 코드를 보면 앞에서 배운 CSS 속성들을 사용한 것을 볼 수 있습니다. **styled-components**를 사용하는 방법과 CSS를 어떻게 작성하는지, 코드의 전체적인 구조를 잘 기억하기 바랍니다.

이제 만든 **Blocks** 컴포넌트를 실제로 화면에 렌더링하기 위해서 **index.js** 파일을 수정해야 합니다. 다음 그림에 표시된 부분을 참고하여 새로 만든 **Blocks** 컴포넌트를 임포트해서 **ReactDOM.render()** 함수에 넣어 주는 코드로 수정합니다.

The screenshot shows the VS Code interface with the file structure on the left and the code editor on the right. The code in index.js imports various components from chapters 03 through 15, including React, ReactDOM, and specific components like Clock, CommentList, NotificationList, Accommodate, Configuration, AttendanceBook, and Signup. It then renders these components within a StrictMode and a root element. A comment at the bottom indicates how to start measuring performance.

```
blocks.jsx index.js
src > index.js
1 Import React from 'react';
2 Import ReactDOM from 'react-dom';
3 Import './index.css';
4 Import App from './App';
5 Import reportWebVitals from './reportWebVitals';
6
7 Import Library from './chapter_03/Library';
8 Import Clock from './chapter_04/Clock';
9 Import CommentList from './chapter_05/CommentList';
10 Import NotificationList from './chapter_06/NotificationList';
11 Import Accommodate from './chapter_07/Accommodate';
12 Import Configuration from './chapter_08/Configuration';
13 Import AttendanceBook from './chapter_09/AttendanceBook';
14 Import Signup from './chapter_11/Signup';
15 Import Calculator from './chapter_12/Calculator';
16 Import ProfileCard from './chapter_13/ProfileCard';
17 Import DarkOrLight from './chapter_14/DarkOrLight';
18 Import Blocks from './chapter_15/Blocks';
19
20 ReactDOM.render(
21   <React.StrictMode>
22     <Blocks />
23   </React.StrictMode>
24   document.getElementById('root')
25 );
26
27 // If you want to start measuring performance in your app, pass a function
28 // to log results (for example: reportWebVitals(console.log))
29 // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
30
31 reportWebVitals();
32
```

▶ 실습 화면 05

코드 작성이 끝났으면 실제로 리액트 애플리케이션을 실행해 보겠습니다. VS Code의 상단 메뉴에서 Terminal > New Terminal을 눌러 새로운 터미널을 하나 실행시킵니다. 이후에 아래 그림처럼 npm start 명령어를 실행합니다.

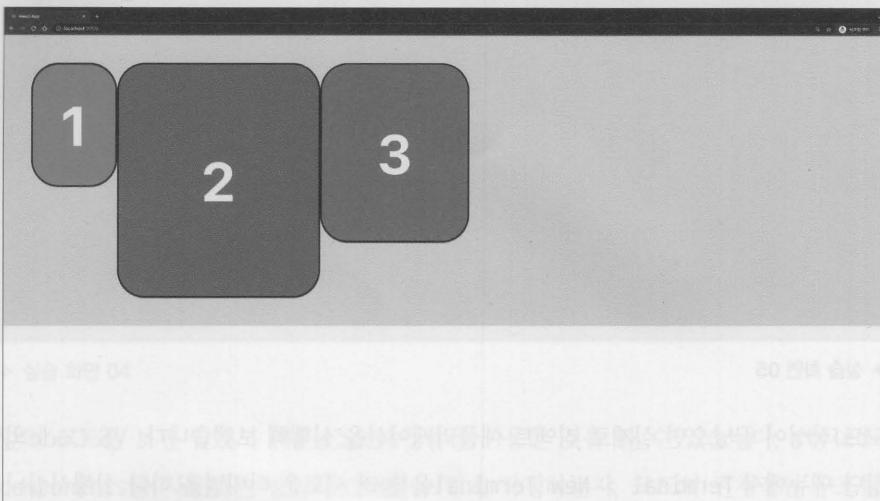
The screenshot shows the VS Code interface with the terminal tab open at the bottom. The terminal window displays the command "npm start" being run, followed by the output message: "The default interactive shell is now zsh. To update your account to use zsh, please run chsh -s /bin/zsh. For more details, please visit: https://support.apple.com/kb/HT208954".

```
blocks.jsx index.js
src > index.js
17 Import ProfileCard from './chapter_13/ProfileCard';
18 Import DarkOrLight from './chapter_14/DarkOrLight';
19 Import Blocks from './chapter_15/Blocks';
20
21 ReactDOM.render(
22   <React.StrictMode>
23     <Blocks />
24   </React.StrictMode>
25   document.getElementById('root')
26 );
27
28 // If you want to start measuring performance in your app, pass a function
29 // to log results (for example: reportWebVitals(console.log))
30 // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
31 reportWebVitals();
32

The default interactive shell is now zsh.
To update your account to use zsh, please run chsh -s /bin/zsh.
For more details, please visit: https://support.apple.com/kb/HT208954.
```

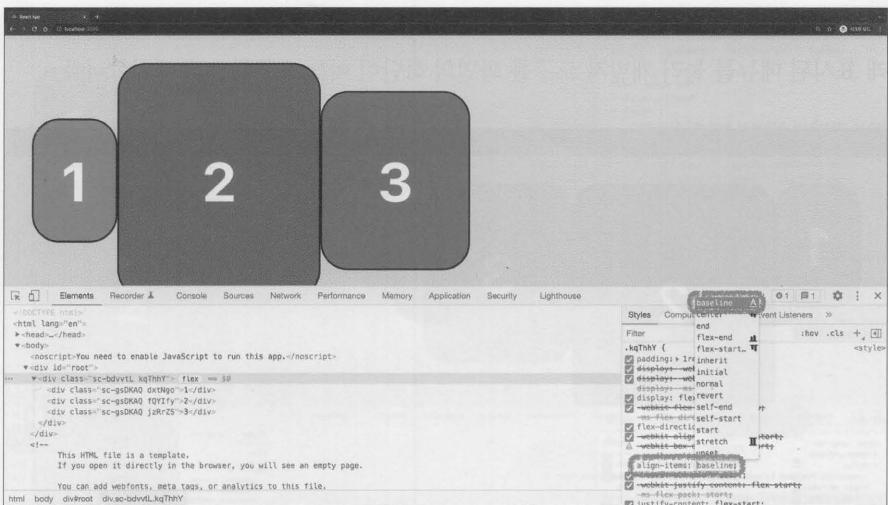
▶ 실습 화면 06

잠시 뒤에 웹브라우저의 새 창이 열리면서 `http://localhost:3000`에 접속되는 것을 볼 수 있습니다. 화면에는 아래 그림처럼 세 개의 박스가 가로로 나열되어 나타나는데 현재 `flex-direction: row`로 되어 있기 때문입니다.



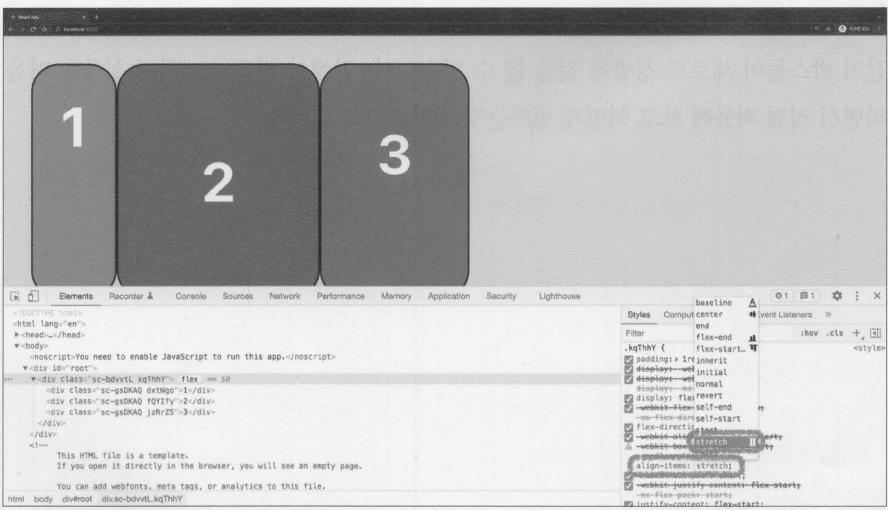
▶ 실습 화면 07

이제 CSS 속성을 동적으로 바꾸면서 화면의 변화를 바로보기 위해서 크롬의 개발자 도구를 열어 엘리먼트 탭을 누릅니다. 그리고 다음 그림과 같이 박스들을 포함하고 있는 DOM 엘리먼트를 찾아 선택하고 오른쪽 패널에서 CSS의 속성을 변경해 봅시다. 여기에서는 `align-items` 속성을 `baseline`으로 바꿨습니다.



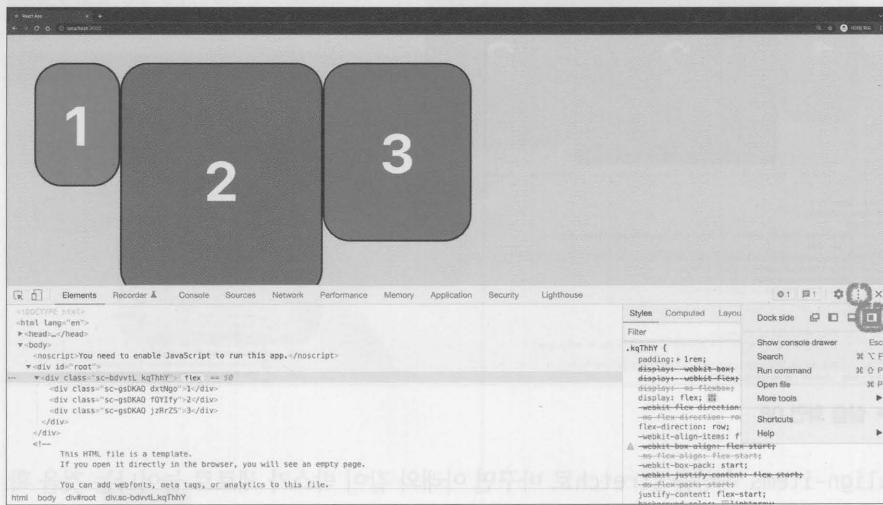
▶ 실습 화면 08

align-items 속성을 **stretch**로 바꾸면 아래와 같이 박스가 세로로 늘어나는 것을 확인할 수 있습니다.



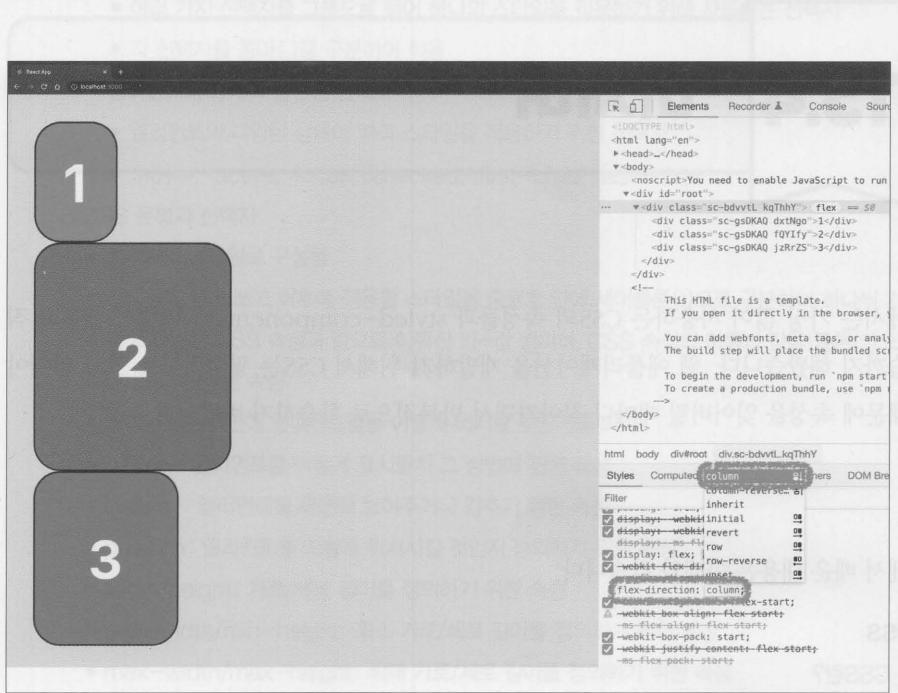
▶ 실습 화면 09

이번에는 박스들을 세로로 정렬해 봅시다. 그전에 개발자 도구를 편하게 보기 위해 아래 표시된 메뉴를 눌러 개발자 도구를 화면의 하단이 아닌 오른쪽으로 배치합니다.



▶ 실습 화면 10

이후 `flex-direction` 속성을 `column`으로 변경해 봅니다. 그리고 나면 다음 그림과 같이 박스들이 세로로 정렬된 것을 볼 수 있습니다. 앞에서 배운 CSS의 속성들을 떠올리면서 직접 적용해 보고 어떻게 변하는지 살펴보기 바랍니다.

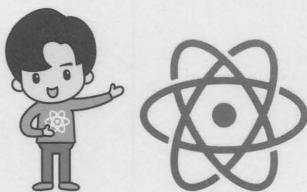


▶ 실습 화면 11

- ▶ 10장에서 배운 CSS의 레이아웃과는 달리 CSS Flexbox는 단일 컨테이너 내에서 항목을 배치하는 데에 초점을 맞추고 있다.
- ▶ 클래스 선택자와 클래스 이름으로 구성되어 클래스는 여러 개의 항목을 관리하는 데에 유용하다.
- ▶ 컨테이너의 클래스와 관련된 CSS 속성은 아래 내용을 통해 살펴보면 좋다.
- ▶ `flex-direction: column;`이 명령어가 대신 `flex-direction: row;`을 사용하는 경우에는 `flex-direction: column-reverse;`과 같은 명령어를 사용하는 경우와 동일한 결과를 얻을 수 있다.
- ▶ `main-axis: flex-direction`은 차례로 흐르는 방향을 지정하는 명령어이다. 예전에는 `flex-direction`이라는 명령어를 사용해도 같은 결과를 얻을 수 있다.
- ▶ `cross-axis: main-axis`를 기준으로 하는 방향을 정하는 명령어이다. 예전에는 `flex-direction`이라는 명령어를 사용해도 같은 결과를 얻을 수 있다.
- ▶ `align-items`
- ▶ 컨테이너 내에서 아이템을 배치하는 방향을 정하는 명령어이다. 예전에는 `flex-align`이라는 명령어를 사용해도 같은 결과를 얻을 수 있다.
- ▶ `cross-align`은 거울로 된 흐름을 정하는 명령어이다. 예전에는 `flex-align`이라는 명령어를 사용해도 같은 결과를 얻을 수 있다.
- ▶ `justify-content`
- ▶ 컨테이너 내에서 아이템을 어떻게 나누는지를 정하는 명령어이다. 예전에는 `flex-justify`이라는 명령어를 사용해도 같은 결과를 얻을 수 있다.
- ▶ `flex-grow`는 컨테이너의 크기로 확장되는 항목을 정하는 명령어이다. 예전에는 `flex-expand`이라는 명령어를 사용해도 같은 결과를 얻을 수 있다.
- ▶ `flex-shrink`은 컨테이너의 크기로 축소되는 항목을 정하는 명령어이다. 예전에는 `flex-collapse`이라는 명령어를 사용해도 같은 결과를 얻을 수 있다.
- ▶ `flex-basis`는 컨테이너의 크기로 초기화되는 항목을 정하는 명령어이다. 예전에는 `flex-initial`이라는 명령어를 사용해도 같은 결과를 얻을 수 있다.



소풀의 처음 만난 리액트



리액트 기초 개념 정리부터 실습까지

경기 과천 교육도서관



EMG345028



무료 동영상
강의